COMP0037 ASSIGNMENT 2

Group: `Group L`

1 Introduction [dai ding]

2 Reactive Planner [Yun]

2.1 Reactive Planning System

A reactive planning system works like that: The planner makes a trajectory using the latest world model and the free space assumption. And the robot constantly perceives the environment and updates the world map as it moves. When the original path has become blocked by an obstacle, it reacts by planning a new set of actions and plans. See the block diagram in Fig. 1.

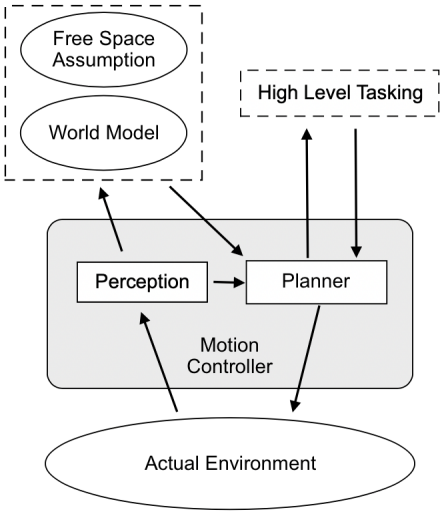


Fig. 1. Reactive Planning System

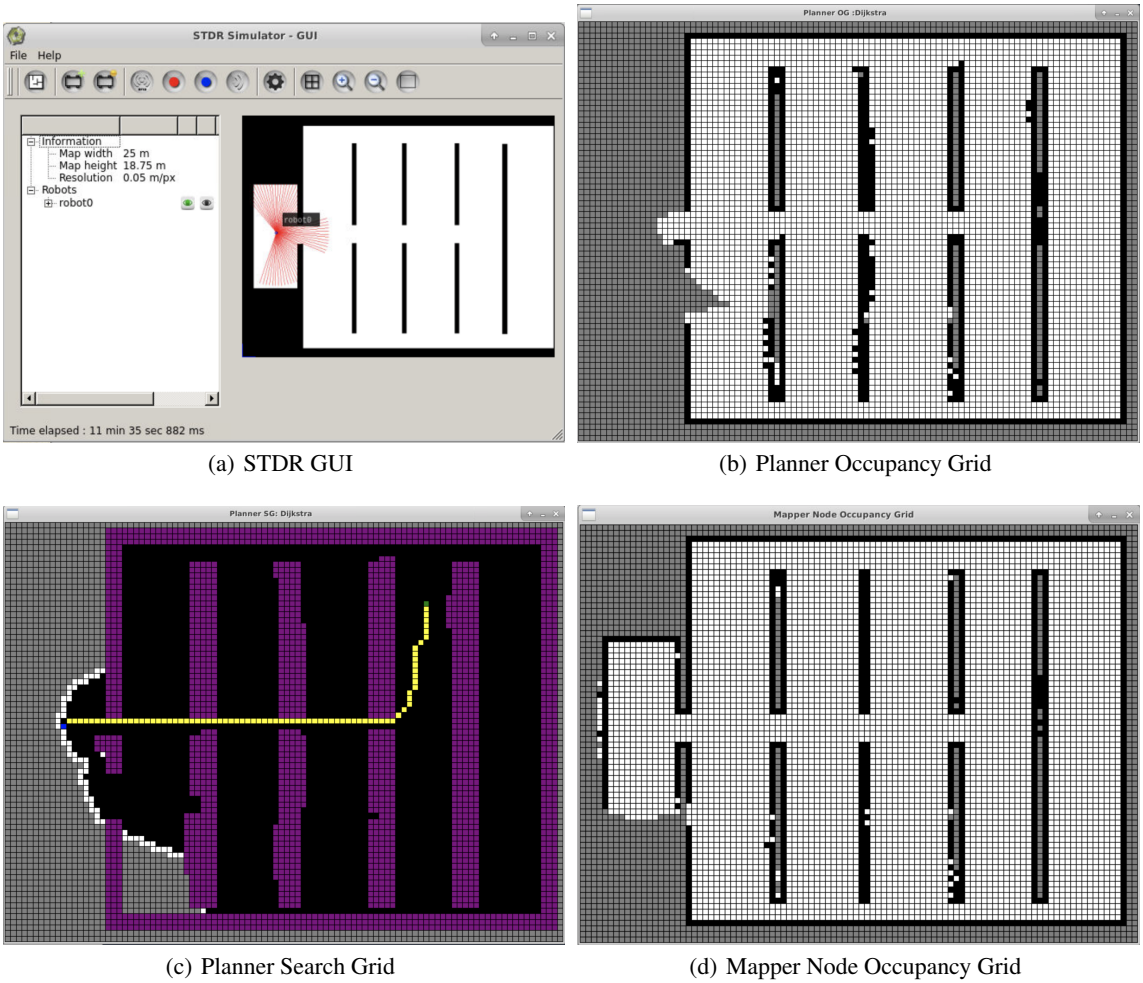
2.2 Our Implementation

We implement the reactive planning system by complete the function *checkIfCurrentPathIsStillGood* in the *ReactivePlannerContoller.py*. The code is shown in Fig. 2.

```
def checkIfPathCurrentPathIsStillGood(self):
    for waypoint in self.currentPlannedPath.waypoints:
        if self.planner.searchGrid.getCellFromCoords(waypoint.coords).label == CellLabel.OBSTRUCTED:
            self.controller.stopDrivingToCurrentGoal()
```

Fig. 2

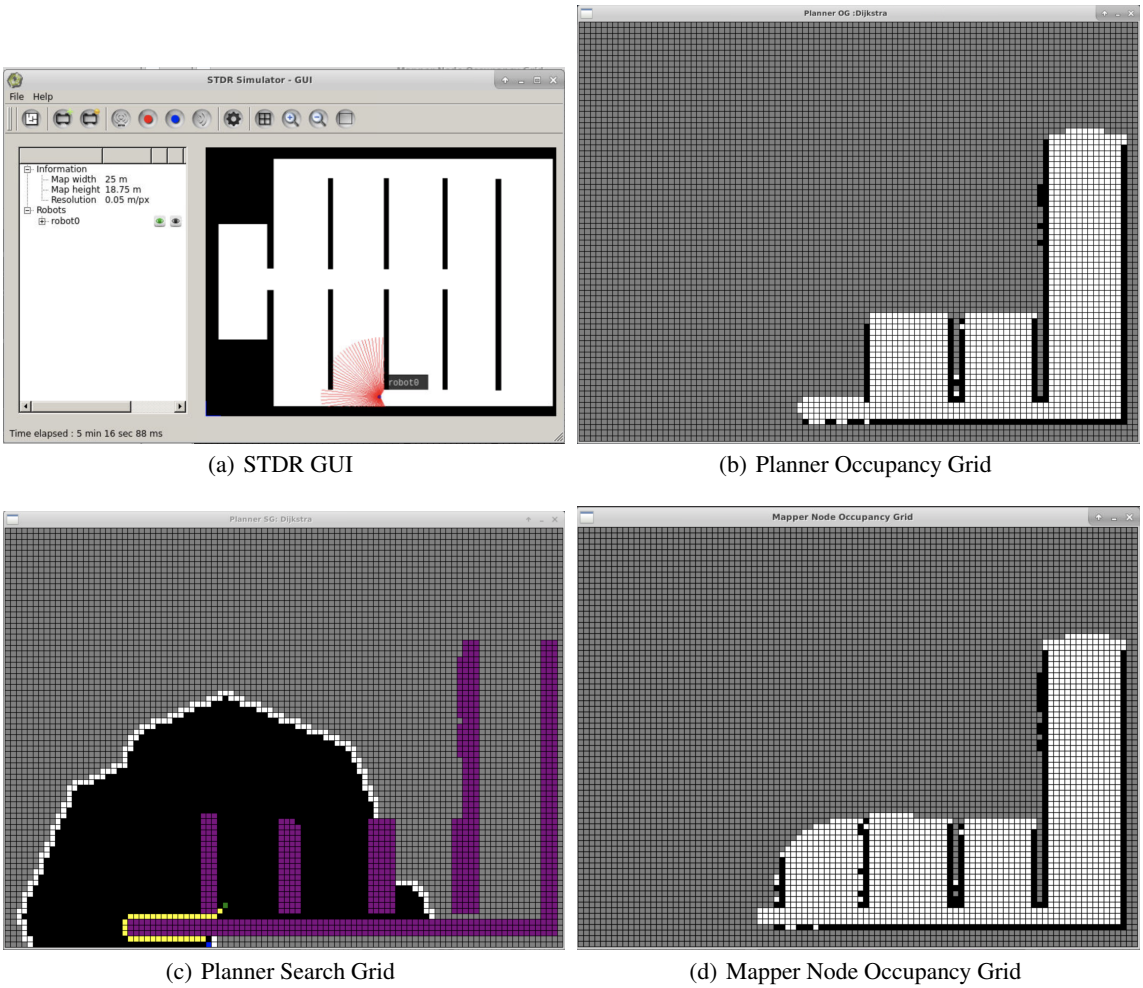
The result is shown in Fig.3 and Fig.4.



(a) STDR GUI (b) Planner Occupancy Grid

(c) Planner Search Grid (d) Mapper Node Occupancy Grid

Fig. 3. Result on the first launch script



(a) STDR GUI (b) Planner Occupancy Grid

(c) Planner Search Grid (d) Mapper Node Occupancy Grid

Fig. 4. Result on the second launch script

2.3 Approach for Improving the Performance

One approach for improving the performance is using a more efficient global planner to reduce the computational cost associated with re-planning. D*-Lite is one such algorithm that caches previous searches and update them only when we need to. It make use of the backwards search. The graph additionally contains a consistency condition. When the graph changes, such as due to an obstacle, the consistency condition is violated. A priority queue is then used to sort the effect of the failed condition on the path. The highest priority updates are committed first.

3 Frontier-Based Exploration System [Yun]

3.1 Frontiers

A frontier is a cell which its state is known while it is adjacent to a cell whose state is not known. Frontier cells define boundary between open space and the uncharted territory. In frontier-based exploration, the robot moves to the boundary to gain the newest information about the world.

Two methods for identifying frontiers are wave front detection and fast frontier detection.The wave front detection explores the map based on the map that has already been explored. It searches the frontiers using depth-first search, starting from the robot initial location. Once a cell is encountered that looks like a frontier, it pauses the search and traces along all the frontier cells. The latter relies on the newly collected sensor information. The sensor data is used to create a contour, which then be separated into frontier and non-frontier segments. The frontier segments are managed in a database to make them persistent. When data becomes available, the frontiers will be split or merged deleted. One

heuristic for choosing next waypoint is picking the closest frontier to the robot. Another one is picking the largest frontier cell.

3.2 The Exploration Algorithm Provided

3.3 Our Implementation

4 Integration of Our Planner and Exploration System [Yun]

5 Information-Theoretic Path Planning [Yusi]

6 Conclusion [dai ding]

References